# Algorithm for computing the partition function of the Potts model for SP-graphs

Sofya Mukhamedzhanova*
Bulat Sabirov†

**Abstract**

The q-state Potts model is a fundamental framework in statistical physics and graph theory, with its partition function encoding rich information about spin configurations. The multivariate Tutte polynomial (known as the partition function of the Potts model) can be defined on an arbitrary finite graph $G$ and encodes a lot of important combinatorial information about the graph. As a special case, it contains the familiar Tutte polynomial with two variables and, consequently, its specialization with one variable, such as the chromatic polynomial, the flow polynomial and the reliability polynomial. The main goal of this paper is to present an efficient algorithm for computing the Potts model partition function on SP-graphs (series-parallel graphs) with arbitrary weights. The algorithm for SP-graphs is based on simplifying the graph by replacing several edges with a single edge of equivalent weight, which significantly reduces computational complexity. In this paper, we present a linear-time algorithm for exactly computing the Potts model partition function on series-parallel graphs (SP-graphs).

## 1 Introduction

The purpose of this paper is to develop and describe an algorithm for calculating the partition function of the Potts model for series-parallel graphs with arbitrary weights. The Potts model is a powerful tool widely used in physics, mathematics, and graph theory. It allows us to model complex systems, including magnetic materials and crystal structures.

*Institute of Mathematics and Mechanics, Kazan Federal University, Kremlevskaya ul. 18, Kazan, 420008 Russia. E-mail: sofyaenglish@gmail.com. Supported by RSF (project No. 24-21-00158).

†Institute of Information technology and Intelligent Systems, Kazan Federal University, Kremlevskaya ul. 18, Kazan, 420008 Russia. E-mail: bulatsabirow@gmail.com.

The multivariate Tutte polynomial, also known as the partition function of the Potts model (see [3]), can be defined for any finite graph and encodes key information about the structure of the graph. Let $G = (V, E)$ be a finite undirected graph, where $V$ is the set of vertices and $E$ is the set of edges. The partition function is defined as follows:

$$Z_G(q, \mathbf{v}) = \sum_{A \subseteq E} q^{K(A)} \prod_{e \in A} v_e, \tag{1}$$

where $q$ and $\mathbf{v} = \{v_e\}_{e \in E}$ are commuting variables, and $K(A)$ is the number of connected components in subgraph $(V, A)$. If we set $v_e = -1$ for all $e \in E$, then $Z_G$ takes the value 1 for each proper coloring of the graph and 0 for each improper one, which allows the computation of the chromatic polynomial [3].

# 2 Preliminaries

The standard definition of a series-parallel graph is: series–parallel graphs are graphs with two distinguished vertices called terminals, formed recursively from $K_2$ by two simple composition operations — series and parallel composition. [4]

In this paper, we provide an alternative definition of series-parallel graphs, equivalent to the standard one, but not starting from $K_2$; instead, it reduces an already existing SP-graph to $K_2$. But first, we introduce two definitions.

**Definition 1.** *Edges $e_1, \ldots, e_n$ are said to be* parallel *if they connect the same pair of vertices $x$ and $y$.*

**Definition 2.** *Edges $e_1, \ldots, e_n$ form a* series connection *between vertices $x$ and $y$ if they form a path in the graph between these vertices and all the internal vertices of the path have degree 2.*

In this paper, by a parallel connection we mean several edges that connect the same pair of vertices (Definition 1), whereas a parallel composition is the graph operation that glues several SP-graphs in parallel at their terminals.

By a series connection, we mean a path of edges all of whose internal vertices have degree 2 (Definition 2), whereas a series composition is the graph operation that glues two SP-graphs in series at their terminals. Now we present an alternative definition of a series-parallel graph, different from the conventional one.

A series-parallel graph is a graph that can be reduced to the graph $K_2$ by alternating the following operations:

1. Replacing all series connections (paths) with a single edge;
2. Replacing all parallel edges with a single edge.

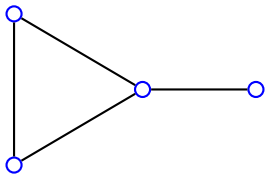


Figure 1: Example of a SP-graph

**Proposition 1.** *[1, 3] If the graph $G$ contains $n$ parallel edges $e_1, \dots, e_n$ connecting the same pair of vertices $x$ and $y$, they can be replaced by a single edge $e$ with weight:*

$$v_e = \prod_{i=1}^{n} (1 + v_{e_i}) - 1. \tag{2}$$

**Proposition 2.** *[1, 3] For series connections of edges $e_1, \dots, e_n$, where there exists a chain of vertices $x_1, \dots, x_{n+1}$ connected in series, one can use the following transformations:*

*1. Replace the sequence of edges $e_1, \dots, e_n$ with a single edge $e = x_1 x_{n+1}$ with weight:*

$$v_e = \frac{q \prod_{i=1}^{n} v_{e_i}}{\prod_{i=1}^{n} (v_{e_i} + q) - \prod_{i=1}^{n} v_{e_i}}. \tag{3}$$

*2. Multiply the partition function of the resulting graph by the prefactor:*

$$\text{pref} = \frac{\prod_{i=1}^{n} (v_{e_i} + q) - \prod_{i=1}^{n} v_{e_i}}{q}. \tag{4}$$

Recall ([1]) that a graph $H$ is called a "necklace" graph if it has two vertices $x$ and $y$, called the "source" and the "sink", respectively, such that all edges of $E(H)$ belong to some series connections between these vertices. Consequently, all the vertices of this graph, except $x$ and $y$, have degree 2 and also belong to series connections between the source and the sink. The different series connections between $x$ and $y$ are called the levels of the "necklace" graph and are numbered 0 to $m$.

**Proposition 3** ([1])**.** *Let $G$ contain a "necklace" subgraph $H$ with arbitrary weights $v_{ij}$, for $i = 0, \dots, m$ and $j = 1, \dots, k_i$, assigned to the edges of the $i$-th level of this "necklace". Suppose that all the vertices of $H$, except the source and the sink, are not incident to any other vertices of the graph $G$. Then the partition function $Z_G$ remains unchanged if the subgraph $H$ is replaced by a single edge $e$, connecting the source and the sink, with weight*

$$v_e = \prod_{i=0}^{m} \left( \frac{\prod_{j=1}^{k_i} (v_{ij} + q) + (q-1) \prod_{j=1}^{k_i} v_{ij}}{\prod_{j=1}^{k_i} (v_{ij} + q) - \prod_{j=1}^{k_i} v_{ij}} \right) - 1, \tag{5}$$

*and the computed partition function is multiplied by the prefactor*

$$\prod_{i=0}^{m} \frac{\prod_{j=1}^{k_i} (v_{ij} + q) - \prod_{j=1}^{k_i} v_{ij}}{q}.$$

Before formulating the proposed algorithm, another idea for its operation was put forward and investigated.

It was that, in order to compute the partition function, one should identify the articulation points of an SP-graph and use them to decompose the original graph into "necklace" subgraphs.

However, this algorithm has several problematic aspects. First, not every SP-graph contains articulation points, so this decomposition is not universally applicable. However, there is a well-known linear - time algorithm [2] for finding all articulation points in graphs. Moreover, finding all simple paths between "source" and "sink" is, in general case, would require a specialized adaptation for the SP-graph class. Each biconnected block with $p$ alternative routes requires enumerating all $p$ simple paths between the terminals of blocks. If such blocks are composed in series, the total number of paths grows as a product of options.

In a typical "moderately parallel" graph, this number grows faster than polynomially, but remains significantly below the worst-case exponential bound. However, if one simply applies the formula for computing the partition function 1, the complexity becomes exponential.

These issues motivated the adoption of a different approach, the use of SP-decomposition trees.

# 3 Algorithm for Finding the Partition Function for Series-Parallel Graphs

An SPQR ("S" stands for "Series", "P" stands for "Parallel", "Q" represents a single edge (a trivial component), "R" stands for "Rigid" (a 3-connected component) ) tree is a tree data structure used in computer science, and more specifically graph algorithms, to represent the triconnected components of a graphs.

The SPQR-tree of a graph may be constructed in linear time and has several applications in dynamic graph algorithms and graph drawing.[5] There are well-known results (see [7]) showing that SP-graphs cannot contain triconnected subgraphs; therefore, the definition of the SPQR-tree for them can be simplified by omitting the R-type nodes. An SPQ-tree, or decomposition tree, of an SP-graph is a binary tree whose internal nodes describe the type of edge connection in the graph (S for series connections and P for parallel connections) and whose leaves (Q) represent the single edges.

An SPQ-tree (hereafter referred to as an SP-tree) can be constructed via edge reduction rules that identify parallel and series connections until the graph is reduced to $K_2$. This process operates in $O(|V| + |E|)$ time and uses $O(|V| + |E|)$ space and can be described as follows [11]:

1. Find all parallel edges in the graph and reduce them using the formula (2). Thus, in the general case, transform a multigraph into a simple graph.

2. Find all vertices with degree 2.

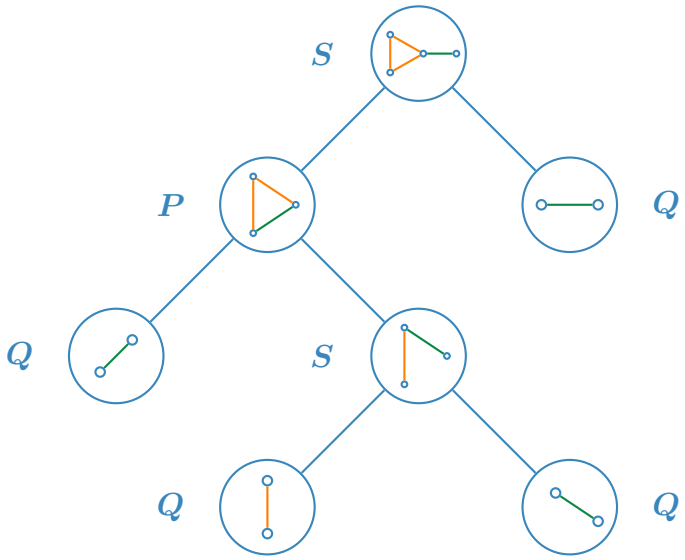



Figure 2: Example of a SP-tree

3. For each such vertices:

   3.1 Retrieve the adjacent vertices.

   3.2 Check if there is an edge between these adjacent vertices.

   - If so, mark it as a parallel edge connection.
   - Otherwise, it is a series connection.

   3.3 Push the current vertex($c$), its adjacent vertices ($a$, $b$), and connection type onto the stack.

   3.4 For each adjacent vertex, check its degree.

   - If its degree is 3, add it to the list of vertices having degree 2.

   3.5 Remove the edges connecting the considered vertex to its adjacent vertices.

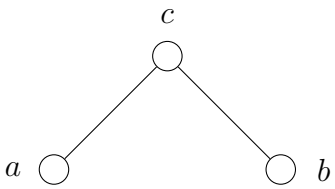
4. For the remaining graph, check the count of vertices with degree 1.

   - If the number of such vertices is not equal to 2, then the input graph is not series-parallel.
   - Otherwise, create a root node that represents the single edge.

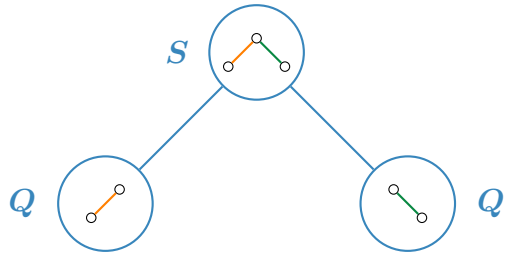
5. Build the SP-tree using stack.

   - For each recorded connection ("series", "parallel" or single egde) and the vertices $c$, $a$, $b$ in the stack, the algorithm retrieves the corresponding node from the partial SP-tree and replaces it with a new internal node representing that connection.

– In the case of a series connection, the SP-tree node corresponding vertices $a, b$ is replaced with an internal node representing a serial structure, where the original path from $a$ to $b$ is subdivided by introducing intermediate nodes, resulting in a path of two connected edges.



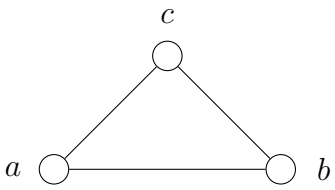

(a) Example of an SP-graph having a series connection



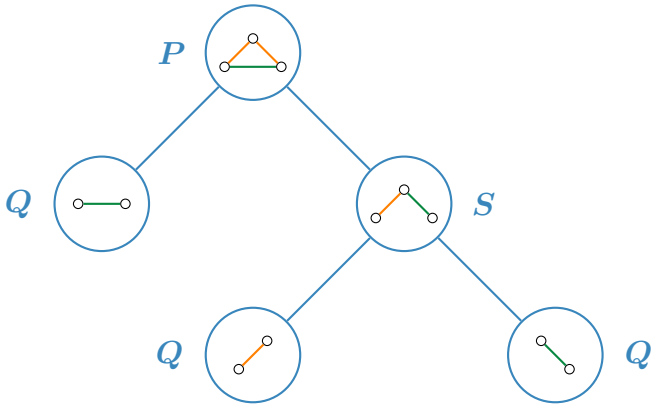

(b) SP-tree corresponding to the SP-graph from (a)

Figure 3: Example for the case of series connection

– In the case of a parallel connection, the node becomes a parallel node with two "children": one representing a direct edge between the "source" and "sink", and the second representing a series connection of two vertices through the intermediate node.




(a) Example of an SP-graph having a parallel connection




(b) SP-tree corresponding to the SP-graph from (a)

Figure 4: Example for the case of parallel connection

This procedure can be summarized in the pseudocode below:

**Algorithm 1:** Build SP-tree

**Input:** SP-graph $G(V, E)$
**Output:** SP-tree rooted at node *root*

$G_1(V_1, E_1) \leftarrow$ `Copied graph` $G$;
$degree2 \leftarrow \{v \mid v \in V_1 : \deg(v) = 2\}$;
$stack \leftarrow \emptyset$;
**while** $degree2 \neq \emptyset$ **do**
  $source \leftarrow$ `pop the element from` $degree2$;
  **if** $source$ *degree is not equal to* 2 **then**
    `Skip cycle iteration`;
  $node_1$ `and` $node_2 \leftarrow$ `neighbors of` source `vertex in` $G_1$ `graph`;
  **if** $(node_1, node_2) \in E_1$ **then**
    `Push onto` $stack$ `parallel composition type`, $source$, $neighbor_1$ `and` $neighbor_2$
  ;
  **else**
    $E_1 \leftarrow E_1 \cup \{(node_1, node_2)\}$;
    $V_1 \leftarrow V_1 \cup \{node_1, node_2\}$;
    `Push onto` $stack$ `series composition type`, $source$, $neighbor_1$ `and` $neighbor_2$
  ;
  **if** $node_1$ *degree equal to* 3 **then**
    $degree2 \leftarrow degree2 \cup \{node_1\}$;
  **if** $node_2$ *degree equal to* 3 **then**
    $degree2 \leftarrow degree2 \cup \{node_2\}$;
  $E_1 \leftarrow E_1 \setminus \{(source, node_1), (source, node_2)\}$;
$degree1 \leftarrow \{v \mid v \in V_1 : \deg(v) = 1\}$;
**if** $|degree1| = 2$ **then**
  $node_1$ `and` $node_2 \leftarrow$ `Vertices in` $degree1$;
  $root \leftarrow$ `Node of SP-tree with source and sink` $node_1$ `and` $node_2$ `respectively`;
**else**
  `raise Error`;
**while** $stack \neq \emptyset$ **do**
  $composition, source, node_1, node_2 \leftarrow$ `Pop the last pushed element from` $stack$;
  `Construct a SP-tree fragment depending on the value of variable` $composition$;

To obtain a weight $w$ for $K_2$ that preserves the Potts partition function of the original

graph, the SP-tree is traversed in a depth-first manner. At each internal node, the "child" weights are combined using the corresponding composition rules, as defined in (3) for the series composition and (2) for the parallel composition. The prefactor (4) introduced by the series composition rule is also taken into account. The depth-first traversal procedure used to compute the effective weight is described in the pseudocode below. Recursively, it visits the SP-tree nodes and applies the appropriate combination rule at each step, depending on the node type.

**Algorithm 2:** Compute equivalent weight on $K_2$ from SP-tree

**Input:** SP-tree rooted at node *root* and constructed from SP-graph $G(V, E)$
**Output:** Equivalent edge weight $w$

$pref \leftarrow 1$;
**Function** `Traverse`(*node*)**:**
  $left \leftarrow$ `Traverse`($node.left$);
  $right \leftarrow$ `Traverse`($node.right$);
  **if** *node.type* `is series connection type` **then**
    $w \leftarrow \frac{q \cdot left \times right}{(left+q)(right+q)-left \times right}$;
    $pref \leftarrow pref \times \frac{left \times right}{weight}$;
  **else if** *node.type* `is parallel connection type` **then**
    $w \leftarrow (1 + left)(1 + right) - 1$;
  **else**
    $w \leftarrow$ `weight of edge` $(node.source, node.sink)$ `in` $G$;
  **return** $w$

For the Potts model in the graph $K_2$, the partition function is given by the simple explicit formula:

$$Z = \left(q^2 + q \times w\right) \times pref \tag{6}$$

# 4 Correctness of the algorithm

To prove the correctness of the algorithm, we show that it transforms any SP-graph into a valid decomposition tree and correctly computes the Potts model partition function. The critical part is ensuring the decomposition tree construction is valid, since the subsequent stages (computing equivalent weights and the final sum) follow standard SP-graph reduction rules and known results. We rely on several properties of series–parallel (SP) graphs:

**Proposition 4.** *In any series-parallel graph (SP-graph), constructed from a single edge using only series and parallel connection, there exists at least one vertex of degree at most 2. In other words, every SP-graph has a vertex of degree 1 or 2.*

*Proof.* SP-graphs are defined recursively. The base case is the graph $K_2$, consisting of two vertices $s$ and $t$ connected by a single edge. If $X$ and $Y$ are SP-graphs with designated terminals $(s_X, t_X)$ and $(s_Y, t_Y)$, then we can create a new SP-graph in two ways:

- **Series composition:** Identify $t_X$ with $s_Y$. The new terminals are $s_X$ and $t_Y$.
- **Parallel composition:** Identify $s_X$ with $s_Y$ and $t_X$ with $t_Y$. The new terminals are the merged pairs.

By recursively applying these operations to copies of $K_2$, we can construct any SP-graph.

Let us prove it by induction.

We prove the proposition by induction on the number of composition operations used in the construction.

**Base case:** In $K_2$, both vertices have degree 1. The claim holds.

**Inductive step:** Suppose any SP-graph constructed in $n$ steps has a vertex of degree $\leq 2$. Let $G$ be an SP-graph constructed in $n+1$ steps via series or parallel composition of $G_1$ and $G_2$.

**Case 1: Series Composition.** Let $G = Series_{composition}(G_1, G_2)$, where $t_1$ from $G_1$ is identified with $s_2$ from $G_2$. Denote this merged vertex by $v$. Then

$$\deg_G(v) = \deg_{G_1}(t_1) + \deg_{G_2}(s_2).$$

By the inductive hypothesis, $G_1$ has a vertex $x$ with $\deg_{G_1}(x) \leq 2$, and similarly for $G_2$. If $x \neq t_1$ and $y \neq s_2$, then both $x$ and $y$ remain valid low-degree vertices in $G$. If $x = t_1$ and $y = s_2$, then $v$ has

$$\deg_G(v) \leq 4,$$

but at least one of the graphs must have another low-degree vertex unless all other vertices have degree $> 2$, which contradicts the inductive assumption. Hence, $G$ contains a vertex of degree $\leq 2$.

**Case 2: Parallel Composition.** Let $G = Parallel_{composition}(G_1, G_2)$ where $s_1 \equiv s_2 = u$ and $t_1 \equiv t_2 = w$. Then

$$\deg_G(u) = \deg_{G_1}(s_1) + \deg_{G_2}(s_2), \quad \deg_G(w) = \deg_{G_1}(t_1) + \deg_{G_2}(t_2).$$

By the inductive hypothesis, both $G_1$ and $G_2$ contain a vertex of degree $\leq 2$. If these are internal vertices, they remain in $G$. If they are terminal, say $x = s_1$ and $y = s_2$, then $u$ has

$$\deg_G(u) = \deg_{G_1}(s_1) + \deg_{G_2}(s_2) \leq 4.$$

Even if $u$ or $w$ has degree up to 4, one of the other vertices still satisfies the degree bound, preserving the claim.

Alternatively, SP-graphs can be reduced to $K_2$ using inverse operations—removal of parallel edges and suppression of degree-2 vertices. Since SP-graphs contain no $K_4$ minors

and are never 3-connected, if all vertices had degree at least 3, reduction to $K_2$ would be impossible. Hence, a vertex of degree $\leq 2$ must always exist.

In all possible constructions of an SP-graph via series and parallel composition, the resulting graph contains a vertex of degree at most 2. This completes the inductive proof. □

**Proposition 5.** *Let $G = (V, E)$ be any simple series–parallel graph constructed from $K_2$ by repeated series and parallel compositions. Then*

$$|E| \leq 2|V| - 3.$$

*Proof.* For planar graphs and bipartite graphs, the relationships between the number of edges and vertices are well known [8]. Although the article [4] contains all the necessary information for deriving an estimate $|E| \leq 2|V| - 3$, but the statement itself and its proof are not given explicitly, so here we will provide a proof of this statement. We proceed by induction on the number of edges $m = |E|$.

**Base case.** For the graph $K_2$, we have $|V| = 2$ and $|E| = 1$. Hence

$$|E| = 1 \leq 2 \cdot 2 - 3 = 1,$$

so the claim holds.

**Inductive step.** Assume the statement holds for all simple SP-graphs with fewer than $m$ edges. Let $G$ be a simple SP-graph with $|E| = m$. Then $G$ is obtained from two smaller SP-graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ by either a series composition or a parallel composition.

*Case 1: Series composition.* In a series composition, one terminal of $G_1$ is identified with one terminal of $G_2$. Thus

$$|V| = |V_1| + |V_2| - 1, \quad |E| = |E_1| + |E_2|.$$

By the inductive hypothesis,

$$|E_1| \leq 2|V_1| - 3, \quad |E_2| \leq 2|V_2| - 3.$$

Therefore

$$|E| = |E_1| + |E_2| \leq (2|V_1| - 3) + (2|V_2| - 3) = 2|V_1| + 2|V_2| - 6.$$

Since $2|V| = 2|V_1| + 2|V_2| - 2$, it follows that

$$|E| \leq 2|V| - 4 < 2|V| - 3,$$

and so the desired inequality holds in this case.

*Case 2: Parallel composition.* In a parallel composition, both terminals of $G_1$ are identified with the corresponding terminals of $G_2$. Hence

$$|V| = |V_1| + |V_2| - 2, \quad |E| = |E_1| + |E_2|.$$

Moreover, since the resulting graph must remain simple, at least one of the graphs, say $G_1$, cannot already have a direct edge between its two terminals (otherwise identification would create a double edge). Consequently $G_1$ has strictly fewer than its maximal allowed edges, and by a refined inductive hypothesis

$$|E_1| \leq 2|V_1| - 4, \quad |E_2| \leq 2|V_2| - 3.$$

Adding these gives

$$|E| = |E_1| + |E_2| \leq (2|V_1| - 4) + (2|V_2| - 3) = 2|V_1| + 2|V_2| - 7 = 2(|V_1| + |V_2| - 2) - 3 = 2|V| - 3,$$

as required.

By induction, the inequality $|E| \leq 2|V| - 3$ holds for all simple SP-graphs. □

**Proposition 6.** *In an SP-graph $G = (V, E)$, if there exists exactly one vertex of degree 2, then there must also exist a vertex of degree 3.*

*Proof.* Assume the contrary: suppose that there exists an SP-graph $G = (V, E)$ such that it has only one vertex of degree 2 (denote it by $v_0$), and all other vertices have degree at least 4.

According to handshaking lemma the sum of degrees of all vertices of a graph is equal to twice the number of its edges:

$$\sum_{v \in V} \deg(v) = 2|E|. \tag{3.2}$$

Now use the previously established inequality 5:

$$|E| \leq 2|V| - 3 \quad \Rightarrow \quad 2|E| \leq 4|V| - 6.$$

Therefore, the total degree satisfies:

$$\sum_{v \in V} \deg(v) = \deg(v_0) + \sum_{v \in V \setminus \{v_0\}} \deg(v) \leq 4|V| - 6.$$

Since all vertices except $v_0$ have degree at least 4, we also have:

$$\sum_{v \in V} \deg(v) \geq \deg(v_0) + 4(|V| - 1) = 2 + 4(|V| - 1) = 4|V| - 2.$$

Combining both estimates, we obtain:

$$4|V| - 2 \leq \sum_{v \in V} \deg(v) \leq 4|V| - 6,$$

which is a contradiction.

Hence, the assumption is false, and the theorem is proved. □

**Proposition 7.** *For decomposition trees built from the same SP-graph, the partition function of the Potts model computed on them is identical.*

As stated in [9] SP-trees constructed for the same SP-graph are equivalent up to the order of series and parallel compositions, then by the Church–Rosser theorem [10] any such tree can be reduced to a single canonical form. Since, according to propositions 1,2, the partition function of the Potts model remains invariant under the reduction of parallel and series edges (with the appropriate weight substituted for the new edge), the values of the partition function computed for any two SP-trees built from the same SP-graph coincide.

The performance data obtained during this work were validated and confirmed by additional experiments that empirically assessed the complexity of the algorithm. To simplify these experiments without sacrificing significant accuracy, the following assumptions were made:

- The parameter $q$, denoting the number of possible states of a vertex in the graph $G(V, E)$, is treated as a constant in the range $[1, 40]$ rather than as a variable.
- The weight of each edge $e$ in $G(V, E)$ is randomly assigned a value from the range $[10^{-5}, 5 \times 10^{-2}]$.
- All statistical outliers in the collected data are excluded from the final dataset used to plot the graph.

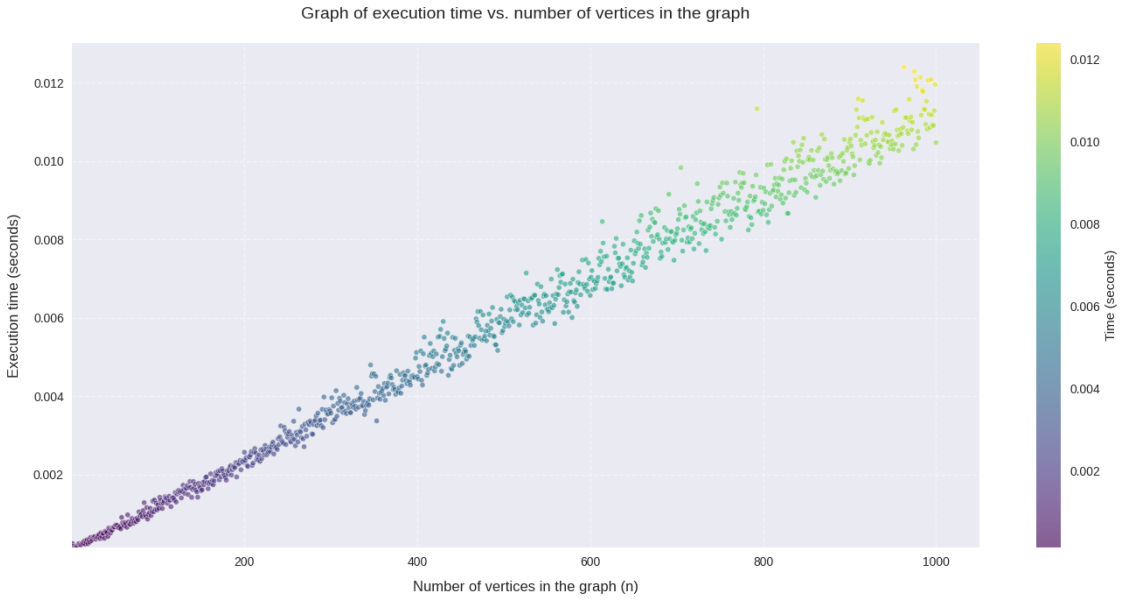



Figure 5: Execution Time of the Proposed Algorithm as a Function of Graph Size

For a clear visual comparison, we also conducted experiments computing the Potts model partition function by its direct definition (see Fig. 6). As the resulting plots demonstrate,

the proposed algorithm for SP-graphs is far more efficient than the classical approach. For example, at $n = 20$ vertices, the traditional method requires about 140 s to produce an answer, whereas the new algorithm takes less than 0.005 s.

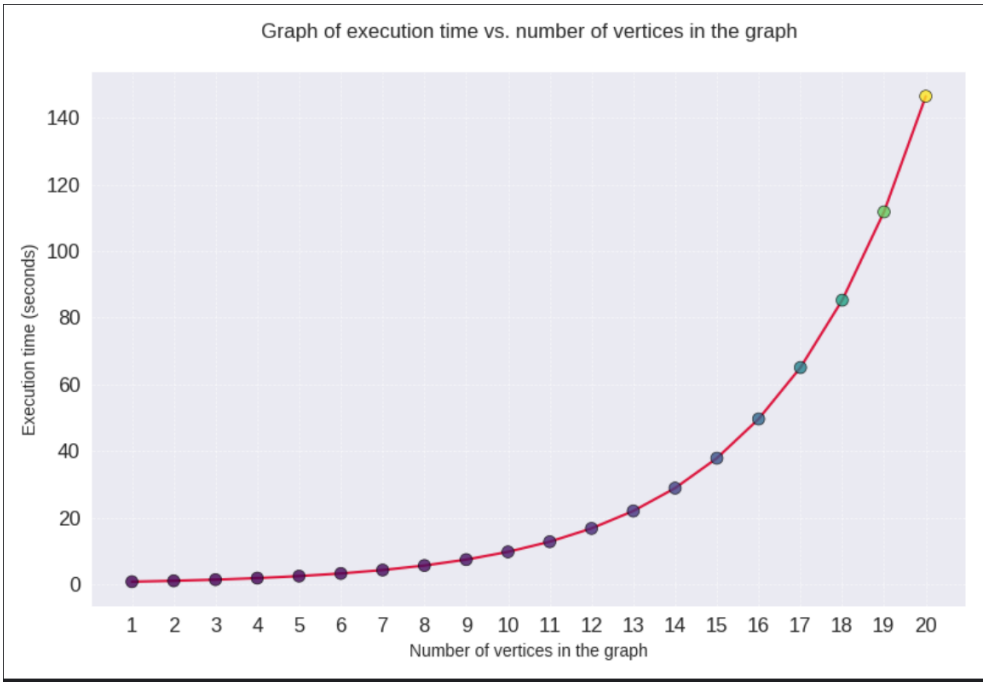

Graph of execution time vs. number of vertices in the graph
Execution time (seconds)
0
20
40
60
80
100
120
140
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Number of vertices in the graph

Figure 6: Inefficient scaling of the direct Potts partition function computation with graph size

# 5 Conclusion

We have developed an efficient linear-time algorithm for computing the partition function of the Potts model (i.e., the multivariate Tutte polynomial) for series-parallel graphs with arbitrary edge weights. The algorithm extends Sokal's series- and parallel-reduction identities [3] and presents SP-graphs as SPQ-trees. By iteratively applying these reductions, the algorithm transforms the original graph into an equivalent two-vertex (single-edge) graph, from which the partition function can be obtained directly. As a result of these simplifications, the computational complexity of evaluating the Potts model partition function on series-parallel graphs using SPQ-trees is significantly reduced to linear complexity.

# References

[1] *Lerner E.Yu., Mukhamedjanova S.A.* Explicit formulas for chromatic polynomials of some series-parallel graphs, *Scientific notes of Kazan University. Series of Physics and Mathematics Sciences*, volume 160, no. 2, pages 339–349. 2018

[2] *Tarjan, R.*, Depth-First Search and Linear Graph Algorithms. SIAM Journal on Computing, 1, 146-160, 1972 https://doi.org/10.1137/0201010

[3] *Sokal A.D.* The multivariate tutte polynomial (alias potts model) for graphs and matroids, Surveys in Combinatorics, volume 327 of London Math. Soc. Lecture Note Series, pages 173–226. Cambridge University Press, Cambridge, 2005.

[4] *Valdes, J., Tarjan, R. E., Lawler, E. L.* The Recognition of Series Parallel Digraphs SIAM Journal on Computing, 1979, Vol. 11, No. 2, pp. 298–313. DOI: 10.1137/0211023

[5] *G. Di Battista and R. Tamassia.* On-Line Maintenance of Triconnected Components with SPQR-Trees. Algorithmica (New York), 15(4), 1996.

[6] *Wilson*, Graph Theory 1736-1936, 2nd edn,. Oxford, 1986

[7] *Brandstädt, Andreas; Le, Van Bang; Spinrad, Jeremy P.* Graph classes: a survey. // SIAM Monographs on Discrete Mathematics and Applications. Vol. 3. – Philadelphia, PA, 1999 – p.172-174

[8] *Diestel, R.* Graph Theory, 5th ed. Springer, 2017

[9] *Valdes Ayesta J.* Parsing Flowcharts and Series-Parallel Graphs. dissertation. 1978.

[10] *Church, A., Rosser, J. B.* Some properties of conversion, Transactions of the American Mathematical Society. — 1936. — V. 39. — № 3. — pp. 472–482.

[11] `https://github.com/ufkapano/graphtheory/blob/master/graphtheory/seriesparallel/sptrees.py`